Support Vector Machines for Broad Area Feature Classification in Remotely Sensed Images

Simon Perkins, Neal R. Harvey, Steven P. Brumby, Kevin Lacker

Space and Remote Sensing Sciences, Los Alamos National Laboratory,
Los Alamos, NM 87545, USA

ABSTRACT

Classification of broad area features in satellite imagery is one of the most important applications of remote sensing. It is often difficult and time-consuming to develop classifiers by hand, so many researchers have turned to techniques from the fields of statistics and machine learning to automatically generate classifiers. Common techniques include maximum likelihood classifiers, neural networks and genetic algorithms. We present a new system called AFREET, which uses a recently developed machine learning paradigm called Support Vector Machines (SVMs). In contrast to other techniques, SVMs offer a solid mathematical foundation that provides a probabilistic guarantee on how well the classifier will generalize to unseen data. In addition the SVM training algorithm is guaranteed to converge to the globally optimal SVM classifier, can learn highly non-linear discrimination functions, copes extremely well with high-dimensional feature spaces (such as hyperspectral data), and scales well to large problem sizes. AFREET combines an SVM with a sophisticated spatio-spectral feature construction mechanism that allows it to classify spectrally ambiguous pixels. We demonstrate the effectiveness of the system by applying AFREET to several broad area classification problems in remote sensing, and provide a comparison with conventional maximum likelihood classification.

Keywords: Support vector machines; feature construction; supervised learning; image feature classification; remote sensing; multispectral imagery; hyperspectral imagery; spatial context

1. INTRODUCTION

1.1. Machine Learning and Remote Sensed Image Classification

Earth-observing satellites produce vast quantities of image data every day, much of it multispectral. These images are used for a wide variety of applications, ranging from weather prediction, through agricultural use monitoring, to making maps of remote areas. One of the core tasks in much of this analysis is the identification in the image of broad area features of interest: clouds, wheat fields, forests, and so on.

Recognition of broad area features can, in many cases, be considered as the problem of performing a pixel-by-pixel classification of a given image. Ideally we want to obtain a *confidence* for each pixel as to whether it belongs to the class of interest, rather than producing a strict binary classification. This allows us to easily trade off the detection rate (DR) against the false alarm rate (FAR) by varying a confidence threshold.

In the remote sensing domain, two main approaches are taken to pixel classification in images: hand-designed; or learned classifiers. Designing a classifier by hand is a slow and laborious process which requires detailed and accurate knowledge of the physics of the object being sought, and of the sensor used to produce the image, and also of the background against which the object will be imaged. If the sensor is upgraded, or its calibration drifts, or if the task requirements are modified even slightly, a redesign of the classifier is usually necessary.

Given the difficulty of hand-designing classifiers, it is natural to look at machine learning and pattern recognition techniques, and ask if they might allow us to generate reliable automatic classifiers much more quickly and easily. Many researchers have examined these approaches, ranging from simple statistical methods such as minimum distance and maximum likelihood classifiers,[1] to more complex approaches such as neural networks[2] and, more recently, Support Vector Machines.[3] While high performance is often achieved with these systems, it is still often or usually the case that a human "eyeballing" the image, visualized in appropriate false colors where appropriate, can do better.

E-mail contact: s.perkins@lanl.gov

1.2. How Do Humans Do It?

One problem with many simple statistical pattern recognition systems that have been developed for pixel-by-pixel image classification is that they base their decisions purely on the spectral information contained in each pixel. Humans, in contrast, have relatively little ability to perceive spectral information, being limited to at most three channels. Instead, they appear to make great use of spatial context and texture information in making decisions.

Clearly, if are going to use machine learning to produce classifiers that can compete with humans, then we need to find a way of incorporating spatial context information into the classifier. One simple way is to provide extra channels for each pixel, each describing some aspect of the local neighborhood. For instance we could apply a set of simple Gaussian smoothing masks to each spectral channel of the image, and then incorporate the smoothed channels as extra elements of the classifier feature vector. These extra channels of information are known as "features" in the pattern recognition community. Instead of smoothing masks we could use sets of other localized filters such as wavelets, Gabor functions, morphological operators, or combinations of these. Texture classification algorithms based on these "filter bank" techniques have been proposed by several researchers.[4,5] Such techniques work well with monospectral images, where the range of textures in the image can be captured well with either a fixed set of filters, or with a simple parameterized class of filters whose parameters can be derived from training data. With multispectral and hyperspectral imagery however, we have to consider the possible spatio-spectral correlations between arbitrary groups of channels and we face an explosion in the number of possible features. How can we decide which features to generate from the infinite variety and number of possible spatio-spectral features?

The first problem is to come up with a consistent and flexible representation that can describe all the possible spatio-spectral features we might want to generate. Human experts, when asked to design an image classification algorithm by hand, typically isolate useful information from the images by applying sequences of standard image processing operations, including smoothing masks, morphological operations and texture operators, and then make a decision using relatively simple thresholds and logic. This process suggests a framework for generating an almost infinite number of possible spatio-spectral features for an image, by chaining together a set of standard image processing operations.

The trick, of course, is to decide which spatio-spectral features to generate. In related work, Draper et al.[6] treat constructing good features as a control problem. They use a reinforcement learning approach to pick primitive operations to chain together. We present a different approach to the same basic problem, inspired by related work in Evolutionary Computation.

2. AFREET

2.1. Motivations

Our current work on AFREET is motivated by the ideas presented above, and by earlier work on a Genetic Programming system for image classification, called GENIE.* Details can be found elsewhere,[8,9] but suffice to say that GENIE uses a fairly standard Genetic Algorithm[10] to evolve a population of image classifiers. Each classifier performs a sequence of primitive image processing steps that transforms the raw image data planes into a set of "answer planes". A linear discriminant, derived by finding the Fisher Discriminant[11] on the training data, is then used to generate a final binary classification for each pixel.

Despite being a relatively unsophisticated algorithm, GENIE produces extremely good classifiers (see, e.g. Harvey et al.[12]). Its success seems to stem from the rich variety of features it is able to construct from the primitive genes, which allows the Fisher Discriminant "backend" to do a good job of classification. However, GENIE makes no explicit attempt to produce classifiers that generalize well, and training times can be very long (many hours). AFREET was developed primarily to address these two problems.

2.2. Design Details

2.2.1. Classifier Structure

Our earlier program, GENIE, works with a population of classifiers. In contrast, AFREET works with a single classifier and attempts to iteratively refine it. This classifier consists of a bank of "feature generators" which transform the raw input image planes into a set of "feature planes". A linear discriminant is then applied on a pixel-by-pixel basis

*Banzhaf et al.[7] provide a good introduction to GP.

to the feature planes to produce a final binary classification plane. The general structure of this classifier is shown in Figure 1. The linear discriminant is described by a weight vector $\mathbf{w}$ and a threshold τ. The "confidence" that a given pixel with feature vector $\mathbf{x}_i$ belongs to the class of interest is given by:

$$c_p = \mathbf{w}.\mathbf{x}_i - \tau$$

A positive confidence indicates that the pixel probably does belong to the class, while a negative confidence indicates that it probably doesn't.

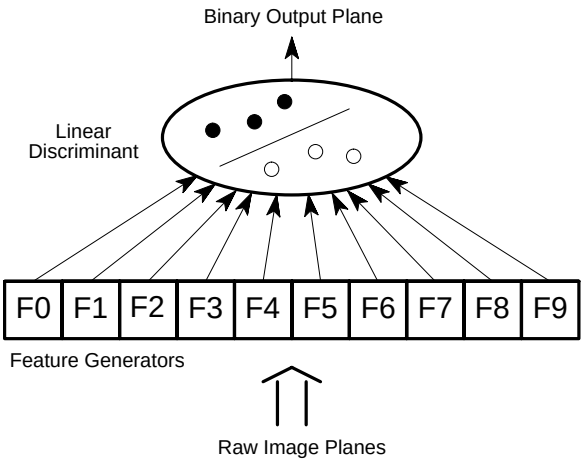

Figure 1. Structure of the classifiers developed by AFREET. The raw image planes are used to derive a number of feature planes, labeled F0 to F9 which are combined using a linear discriminant to give a final binary classification.

Each feature generator is represented as a program tree, a representation commonly used in genetic programming systems. The primitive operations that are available in Afreet are listed in Table 1. All of the operators take zero or more images as input, and produce a single image as output. Figure 2 shows a typical tree that might generate one of the feature planes in Figure 1.

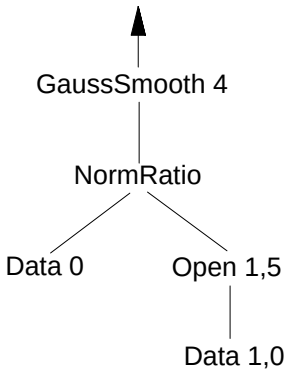

Figure 2. A typical feature generator used to generate one feature plane. This particular generator performs a morphological opening on plane 1 of the input image, using a linear structuring element of radius 5. It then finds the normalized ratio between this result and input image plane 0. Finally this ratio image is smoothed with a Gaussian mask of radius 4.

Before the feature planes are generated, the pixel values in the input image are rescaled so that the minimum value is 0.0 and the maximum value is 1.0. All operators used in AFREET assume that input pixel values are of the

Table 1. Primitive operations used to construct AFREET features. Data is the basic operation used to access the input image. All other operations are neighborhood operations, except Peak and NormRatio, which work on single pixels. Open and Close may use a linear structuring element. The relevant morphological operation is carried out with this S.E. at all possible distinct orientations on the discrete pixel grid, and the output value is the maximum of all possible openings, or the minimum of all possible closings, as appropriate. Consult any good image processing textbook for more details. In the descriptions below, the radius parameter can take values between 1 and 10; the center parameter takes a value between 0.0 and 1.0; and the shape parameter can takes values of DISK or LINE.

NAME	INPUTS	PARAMS	DESCRIPTION
Data	0	index, scale	Extracts raw data plane index from the input image sub-sampled by a factor 2^{scale}.
GaussSmooth	1	radius	Gaussian smoothing with a kernel of the specified radius.
Grad	1	radius	Smoothed gradient magnitude using Gaussian smoothing with a kernel of the specified radius.
Min	1	radius	Minimum value within radius pixels of each pixel
Max	1	radius	Maximum value within radius pixels of each pixel
StdDev	1	radius	Standard deviation in circular neighborhood of given radius.
Peak	1	center	Non-linear transfer function. Pixel values are mapped onto new values given by a Gaussian function with the given center and standard deviation 0.25.
Open	1	shape, radius	Morphological opening. shape determines if a disk or linear structuring element is used. radius determines the size of the S.E.
Close	1	shape, radius	Morphological closing. Parameters as for Open.
WTopHat	1	shape, radius	Performs a morphological opening as above, but then subtracts the result from the original image.
BTopHat	1	shape, radius	Performs a morphological closing as above, but then subtracts the original image from the result.
NormRatio	2	—	Normalized ratio between two planes: $(\frac{a-b}{a+b}+1)\times 0.5$

order of unity and ≥ 0, and they produce output which has the same properties.[†] Once a feature plane is generated it is then rescaled again to have a mean value of 0.0 and a standard deviation of 1.0. The various normalization scales and offsets are only calculated during training. When the trained classifier is being applied to new images, the previously derived normalization values are re-used.

2.2.2. Initialization

The set of feature generators is generated randomly at the start of training subject to two constraints: (a) the feature set contains no duplicate generators, and (b) the depth of the generators does not exceed a user-defined limit, typically taken to be 3. The smallest possible tree has a depth of 1. We use a generation technique that encourages compact trees, by linearly increasing the probability of a terminal node being selected towards 1.0, as the depth limit is approached.

2.2.3. Training the Discriminant

The linear discriminant is trained as a Support Vector Machine,[13,14] using a modified version of Platt's SMO algorithm,[15] suggested by Keerthi et al.[16] An SVM is essentially a linear discriminant that satisfies certain constraints that are designed to guarantee a good level of generalization ability on out of sample data. SVMs belong to a class of classifiers known as "large margin classifiers". These classifiers not only attempt to separate pixels into one category or another, but to do so with as much "margin" as possible. For data that is perfectly separable, the margin is simply defined as the distance from the discriminant hyperplane to the nearest data point. Intuitively, maximizing this distance gives a more robust classifier. SVMs thus avoid the overfitting problems found with many other classifiers and scale well to very high dimensional data.

[†]This accounts for the slightly strange formula for the NormRatio operator.

One of the nice things about SVMs apart from their generalization properties is that they can be used to perform highly non-linear classification through the use of a "kernel". For AFREET, we do not use this advantage, since it would interfere with our method of feature selection, described below. The SVMs used by AFREET are "linear SVMs".

Our training data consists of training images, in which pixels have been marked as belonging to a "positive" class ($y_i = +1$), or a "negative" class ($y_i = -1$). In general there will be different numbers of pixels in each class. We have found in our applications that better results are often achieved if we try equally hard to get both categories correct, rather than simply minimizing total misclassifications. Therefore we use a cost function that makes the total importance of the "positive" pixels equal to the total importance of the "negative" pixels. This is easily done by modifying the usual SVM objective function to be optimized to the following:

$$\frac{1}{2}\|\mathbf{w}\|^2 + \sum_i C_i \xi_i \quad \begin{cases} C_i = \frac{K}{n_+} & \text{if } y_i = +1 \\ C_i = \frac{K}{n_-} & \text{if } y_i = -1 \end{cases} \tag{1}$$

where ξ_i are the "slack variables", $\mathbf{w}$ is the vector of weights describing the linear discriminant, K is a constant, y_i is the class label associated with the ith pixel, and n_+ and n_- are the numbers of pixels in the positive and negative classes respectively. In the more usual SVM formulation, a single constant C is used for all pixels. See Burges[14] for more details.

2.2.4. Evolving the Feature Set

It is unlikely that the randomly generated initial feature set constitutes an ideal basis for classification, so AFREET attempts to iteratively refine the features using the algorithm shown in Figure 3. The algorithm essentially involves deciding heuristically which features are the most important and least important, and then either replacing the least important with randomly generated new features, or replacing the most important with small "mutations" of that feature. If the change produces a significant decrease in the value of the SVM objective function, then it is kept, otherwise we revert back to the unmodified feature. If the objective function is essentially unchanged (to within 1%), then the new feature is kept if it is computationally less expensive than the old one. This refinement strategy is essentially a greedy one. For the feature refinement process, we save time by only training on a randomly chosen subset of all the training points, typically of size 10,000. Once a feature set has been chosen, all the points are used for the final optimization.

```
Randomly initialize features
Perform initial optimization
for j = 1 to F:
    let p_m = j/F
    let T = ln((1-p_a)/2) / ln((n-1)/n)
    Randomly choose T feature indices I_1 ... I_T, with replacement
    with probability p_m:
        Find i such that |w_{I_i}| is maximized
        Mutate feature with index I_i
    else:
        Find i such that |w_{I_i}| is minimized
        Randomize feature with index I_i
    Re-optimize
    If objective function decreased, then keep change, otherwise revert
end for
```

Figure 3. The feature set refinement algorithm. F is the desired number of refinement cycles; p_m is the probability of mutating a good feature, rather than randomizing a bad one; T is the tournament size; n is the size of the current feature set; p_a is the desired probability of picking the absolute best or worst feature in the tournament; w_k is the component of the weight vector associated with the kth feature.

The importance of a feature is decided simply by looking at the magnitude of the weight vector component associated with that feature. Since all features are normalized to have the same mean and standard deviation, this component provides some indication of how important a particular feature is. Note that we only have direct access

to the weight vector for linear SVMs. Linear SVMs are also preferred because training times are much faster using SMO. We use an evolutionary computing method called "tournament selection" to select the "best" and "worst" feature planes, which introduces some stochasticity into the selection process. The tournament size is based on the current feature set size, and is chosen to give a fixed probability (typically 0.25) of choosing the absolute best or worst feature.

Three kinds of mutations are employed with equal probability: (a) **Parameter Mutation**: A single node with a parameter is picked, and that parameter is mutated in a manner appropriate to the parameter type. (b) **Grow Mutation**: A new node is generated randomly and inserted at the head of the generator tree. The old tree becomes one of its arguments. If the new node has more than one argument, the other arguments are generated randomly. (c) **Shrink Mutation**: The opposite of Grow: The head node is removed, and one of its arguments is selected randomly to be the new feature tree.

These mutations are partially inspired by the ways in which a human would typically modify an image processing pipeline, while experimenting with possibilities, but are obviously less well directed. Mutations that would cause the tree depth to exceed a user-defined depth limit (greater than the initialization depth limit) are not allowed. This absolute depth limit is typically set to 5.

The probability of a mutation is adjusted gradually and linearly from 0 at the beginning of the refinement process to 1 at the end. The motivation for this is that at the beginning of the run it is more important to get rid of useless features and try random replacements, while towards the end, we hopefully have quite a good set of features and so should focus on improving them.

2.2.5. Pruning

One of the potential advantages of a population-based approach such as GENIE is that it begins by considering many different solutions, and then as the training run progresses, it tends to converge attention on a much smaller number of solutions. We can obtain some of this effect in AFREET by starting with an initial feature set that is much larger than the feature set we want to end up with. In conjunction with the refinement process, we perform a series of pruning steps, each of which simply eliminates the feature with the current lowest weight component and re-optimizes. Typically when using pruning, we start with a feature set ten times larger than the final feature set, and prune down to the final set size during the initial 50% of the feature refinement process.

2.2.6. Interface Issues

Obtaining accurate training data is an essential part of a learning system. We use a Java GUI called ALADDIN to allow human experts to provide training data. ALADDIN allows the user to visualize a multispectral image, and "paint" regions where the feature of interest is found, and also regions where it is not found. Not all pixels need to be classified. The same interface is used to visualize classification results and can be used to refine the training data before re-training.

3. EXPERIMENTS

To test and illustrate the effectiveness of AFREET we include some examples of the system being applied to broad area classification problems in several different types of multispectral and hyperspectral imagery.

3.1. Data Sets

Figure 4 shows the training and test data that were used. Three tasks were selected:

1. Finding sandy beaches in 3 channel color USGS Digital Orthoquad aerial photography of Cape Kennedy, US. This task is difficult because with the limited number of spectral channels available, the beach in our training images is spectrally identical to many non-beach areas of the image. The images we use have been sub-sampled by a factor of 2 from the original data (pixels are approx. 2 m square).

2. Finding urban areas in 224 channel AVIRIS images of the Moffet field area in the US. A difficult task owing to the rather subjective nature of what defines urban area, and because of the variability of materials found in urban areas. The hyperspectral data also tests the ability of the feature refinement process to search an extremely large space of possible features.

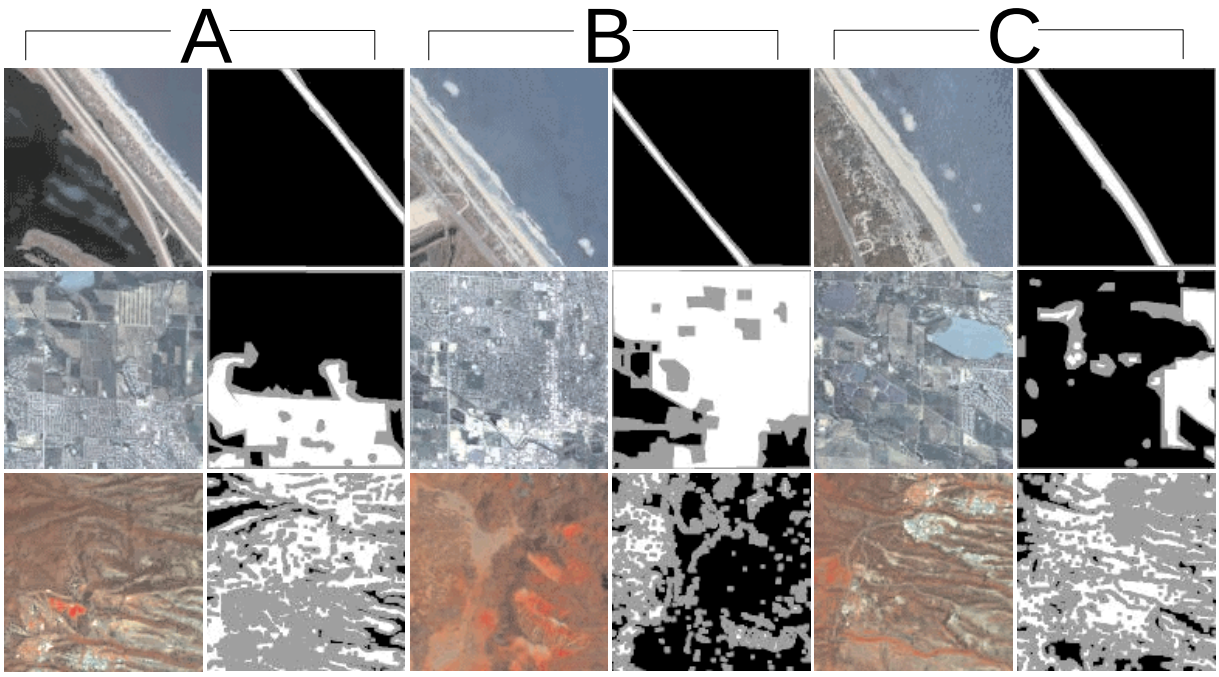

Figure 4. Training data. Each row shows the training/test images used for a different task. From top to bottom: beach, urban and pine forest. To the right of each scene is shown the ground truth associated with that scene. White corresponds to the feature of interest being present, black corresponds to the feature not being present, and gray means no ground truth was available for that pixel. The three scenes for each task were used for three-fold cross-validation, as explained in the text.

3. Finding Ponderosa Pine forest in Landsat 7 ETM+ images. We used bands 1–5 and 7 from imagery taken of northern New Mexico (Path 34, Row 35, July 1st, 1999). The hard part of this task is distinguishing the Ponderosa Pine forest from the neighboring mixed conifer forest, which is spectrally very similar.

For the first two tasks, "ground truth" was created by a human analyst marking up the images by eye. In the high resolution imagery used for these two tasks, the features of interest are easily seen. For the third task, ground truth was provided by S.W. Koch of Los Alamos National Laboratory. The ground truth was derived using unsupervised k-means clustering of a Landsat 5 TM image, with clusters being identified using a ground campaign and categorized into 12 different land cover classes. Before using this ground truth here, we eroded the boundaries of each land cover class to reduce problems caused by slight mis-registration of the ground truth and underlying imagery.

3.2. Experimental Details

Experiments were performed to compare AFREET with the commonly used maximum likelihood classifier. The ML classifier was given only spectral information for each pixel. This could be considered a slightly unfair comparison, since AFREET has access to spatial context through its feature generation process, but, at least for the 224 channel hyperspectral data, there was no good tractable alternative. Our purpose here is simply to demonstrate that AFREET is capable of constructing good spatio-spectral features on its own.

The AFREET experiments used a final feature set of size 10, pruned down from an initial set of 100 randomly generated features. The feature refinement process was run for 100 cycles, and the initial feature set was pruned down to the final size over the first 50 cycles. For the beach and forest finding tasks, we used $K = 500$ in the SVM objective function (expression 1). For the somewhat harder urban finding task, we used $K = 50$ to reduce training times at the expense of trying less hard to match the training data perfectly.

For each task, three 256x256 images were selected and labeled. We tested our classifiers using three-fold cross-validation. Each combination of two out of the three images was used to train a different classifier, and the resulting

Table 2. Training and test scores for the two classifiers (A: AFREET, B: maximum likelihood) and each of the three tasks. The table shows the detection rate, false alarm rate and the average misclassification rate. The uncertainties shown are the standard error of the mean over all runs on each task.

TRAINING SCORES									
	BEACH			URBAN			PINE FOREST		
EXP	DR %	FAR %	MISS %	DR %	FAR %	MISS %	DR %	FAR %	MISS %
A	100.0 ± 0.002	0.32 ± 0.09	0.16 ± 0.05	98.40 ± 0.20	1.56 ± 0.22	1.58 ± 0.14	98.83 ± 0.14	1.99 ± 0.24	1.58 ± 0.19
B	99.27 ± 0.23	3.69 ± 0.15	2.21 ± 0.04	82.35 ± 12.34	2.72 ± 1.37	10.19 ± 5.66	85.45 ± 6.52	4.44 ± 1.89	9.50 ± 2.36
TEST SCORES									
	BEACH			URBAN			PINE FOREST		
EXP	DR %	FAR %	MISS %	DR %	FAR %	MISS %	DR %	FAR %	MISS %
A	99.54 ± 0.12	1.79 ± 0.98	1.13 ± 0.47	92.69 ± 3.33	9.10 ± 3.91	8.21 ± 2.05	94.97 ± 0.66	25.50 ± 3.97	15.27 ± 1.86
B	99.61 ± 0.08	4.18 ± 0.31	2.28 ± 0.14	92.18 ± 3.95	19.87 ± 14.06	13.85 ± 5.17	83.07 ± 10.15	21.91 ± 13.74	19.42 ± 3.41

classifiers were each tested on the third out of training sample image. For AFREET each of these training combinations was repeated three times with different random number seeds to reduce the effect of random fluctuations. The performance scores reported in the results section are the mean scores over all the runs for a given classifier type and task.

4. RESULTS

Table 2 shows the experimental results. The detection rate gives the percentage of pixels that are really in the positive class that were misclassified. The false alarm rate gives the percentage of pixels in the negative class that were misclassified. The average misclassification rate gives the mean of FAR and $(100 - \text{DR})$. Due to the weighting scheme described in 2.2.3, it is this average that is optimized during AFREET training. Both training and test scores are shown.

The general pattern of these results shows that AFREET outperforms both the purely spectral maximum likelihood classifier, on both training scores and test scores. This indicates that it is able to successfully construct useful spatio-spectral features that allow it to solve the classification task at hand.

Figure 5 compares the pixel classifications produced by AFREET, with the results from maximum likelihood classification. This shows clearly the power of using spatial context information in order to classify spectrally ambiguous image features.

5. CONCLUSIONS AND FURTHER WORK

For many real-world image classification problems, purely spectral feature vectors are not sufficient, yet choosing a good set of spatial context features for multispectral and hyperspectral data is very difficult. We have presented AFREET, an SVM-based system that attempts to automatically construct spatio-spectral feature vectors by putting together image processing operators to form pipelines similar to those hand-designed by humans. Ideas from evolutionary computing are used to refine the feature set from a random initial selection. In the experiments described here, AFREET performs a good job of classification, and clearly uses spatial context to good advantage to outperform purely spectral classifiers in most cases.

One important thing that AFREET lacks is a sense of direction when trying to mutate and replace feature generators. Humans do not make totally random changes when optimizing image processing pipelines — they make purposeful changes. We are currently examining a boosting technique that will allow AFREET to rapidly estimate the effect of adding in a different feature generator without having to re-optimize. This will allow AFREET to screen a much larger set of potential features in a shorter time, and to make refinements that are more sensible.

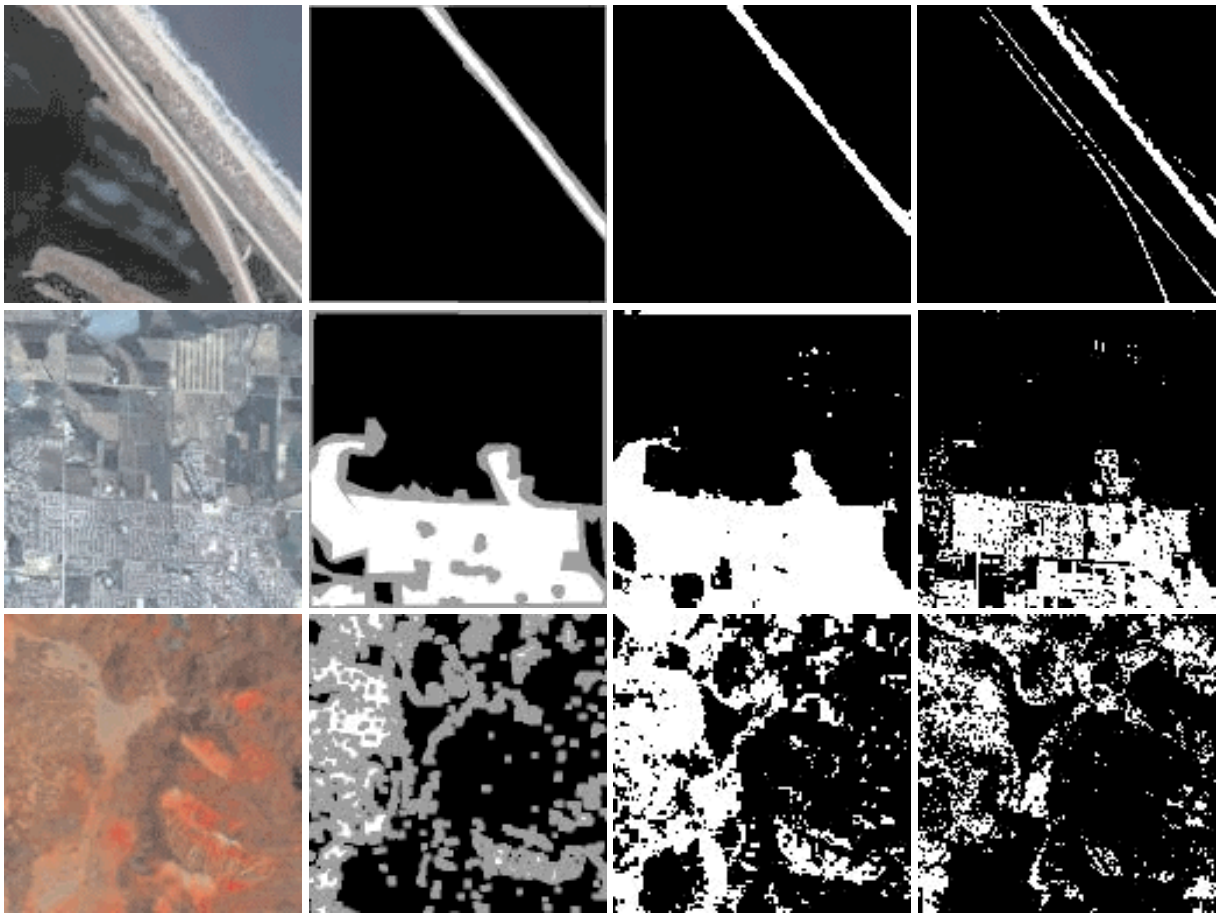

Figure 5. Result images from Experiment 4 for each task. Each row shows a different task. From top to bottom: beach, urban and pine forest. The first column shows an example test image, the second column shows the ground truth provided for that image, the third column shows the output of an AFREET classifier trained on the other two images for that task (i.e. the image shown was not included in the training sample), and the third column shows the equivalent result for a maximum likelihood classifier. The three different AFREET runs performed for each training file combination produce slightly different output images. The one shown is for the classifier that obtained the highest training score.

ACKNOWLEDGMENTS

Many of the ideas in AFREET were inspired by earlier work on the GENIE system referred to in the text. GENIE was developed by Jeff Bloch, Steven Brumby, Reid Porter, John Szymanski, James Theiler, Cody Young and the authors of this paper. This work was supported by the US Departments of Defense and Energy.

REFERENCES

1. J. Richards, *Remote Sensing Digital Image Analysis*, Springer-Verlag, 1993.
2. H. Bischof and A. Leonardis, "Finding optimal neural networks for land use classification," *IEEE Transactions on Geoscience and Remote Sensing* **36**(1), pp. 337–341, 1998.
3. F. Roli and G. Fumera, "Support vector machines for remote-sensing image classification," in *Proc. EOS/SPIE Symposium*, (Barcelona), Sept. 2000.
4. A. Bovik, M. Clark, and W. Geisler, "Multichannel texture analysis using localized spatial filters," *IEEE Trans. Pattern Analysis and Machine Intelligence* **12**, pp. 55–73, 1990.

5. A. Jain and F. Farrokhina, "Unsupervised texture segmentation using gabor filters," *Pattern Recognition* **23**, pp. 1167–1186, 1991.
6. B. Draper, J. Bins, and K. Baek, "ADORE: Adaptive object recognition," in *Proc. International Conference on Vision Systems*, pp. 522–537, (Las Palmas de Gran Canaria, Spain), Jan. 1999.
7. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann, San Francisco, CA, 1998.
8. S. Perkins, J. Theiler, S. Brumby, N. Harvey, R. Porter, J. Szymanski, and J. Bloch, "GENIE: A hybrid genetic algorithm for feature classification in multi-spectral images," in *In Proc. SPIE 4120: Applications and Science of Neural Networks, Fuzzy Systems and Evolutionary Computation III*, 2000.
9. J. Theiler, N. Harvey, S. Brumby, J. Szymanski, S. Alferink, S.Perkins, R. Porter, and J. Bloch, "Evolving retrieval algorithms with a genetic programming scheme," in *Proc. SPIE 3753*, pp. 416–425, 1999.
10. J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
11. C. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.
12. N. Harvey, S. Perkins, S. Brumby, J. Theiler, R. Porter, A. Young, A. Varghese, J. Szymanski, and J. Bloch, "Finding golf courses: The ultra high tech approach," in *Real World Applications of Evolutionary Computing*, S. C. et al., ed., vol. 1803 of *Lecture Notes in Computer Science*, Springer, 2000.
13. V. Vapnik, *The Nature of Statistical Learning Theory*, Springer, NY, 1995.
14. C. Burges, "A tutorial on support vector machines for pattern recognition," *Knowledge Discovery and Data Mining* **2**(2), 1998.
15. J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, eds., pp. 185–208, MIT Press, 1999.
16. S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murphy, "Improvements to platt's smo algorithm for svm classifier design," Tech. Rep. CD-99-14, Dept. of CSA, IISc, Bangalore, India, 1999.